I/O and Lustre: An Application Programmer's Perspective

Bilel Hadri and Lonnie Crosby
bhadri@utk.edu lcrosby1@utk.edu
NICS Scientific Computing Group

OLCF/NICS Spring Training

March 9th, 2011

Outline

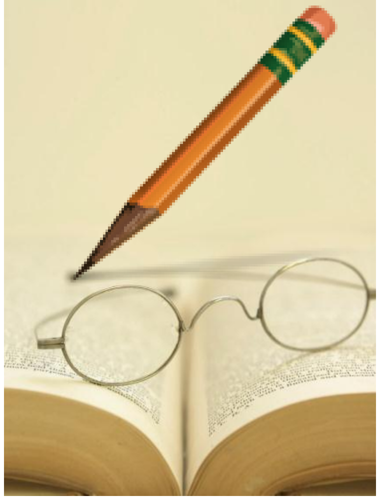
- Introduction to I/O
- Path from Application to File System
- Common I/O Considerations
- I/O Best Practices

Outline

- Introduction to I/O
- Path from Application to File System
- Common I/O Considerations
- I/O Best Practices

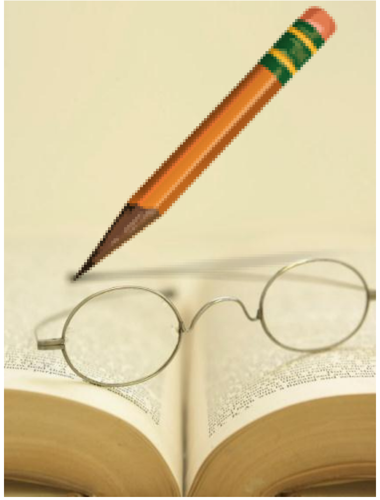
Factors which affect I/O.

- I/O is simply data migration.
  – Memory ⟷ Disk
- I/O is a very expensive operation.
  – Interactions with data in memory and on disk.
- How is I/O performed?
  – I/O Pattern
    - Number of processes and files.
    - Characteristics of file access.
- Where is I/O performed?
  – Characteristics of the computational system.
  – Characteristics of the file system.



I/O Performance

- There is no "One Size Fits All" solution to the I/O problem.
- Many I/O patterns work well for some range of parameters.
- Bottlenecks in performance can occur in many locations. (Application and/or File system)
- Going to extremes with an I/O pattern will typically lead to problems.

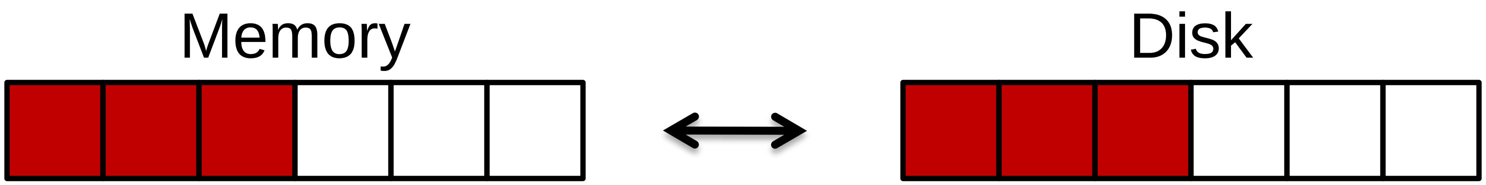


Outline

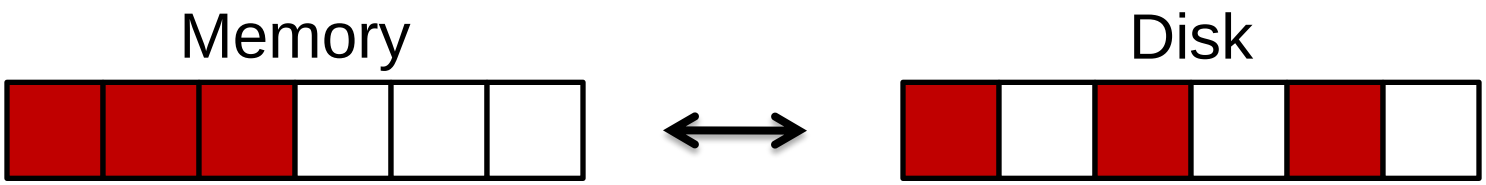
- Introduction to I/O
- Path from Application to File System
  - Data and Performance
  - I/O Patterns
  - Lustre File System
  - I/O Performance Results
- Common I/O Considerations
- I/O Best Practices

Data and Performance

- The best performance comes from situations when the data is accessed contiguously in memory and on disk.

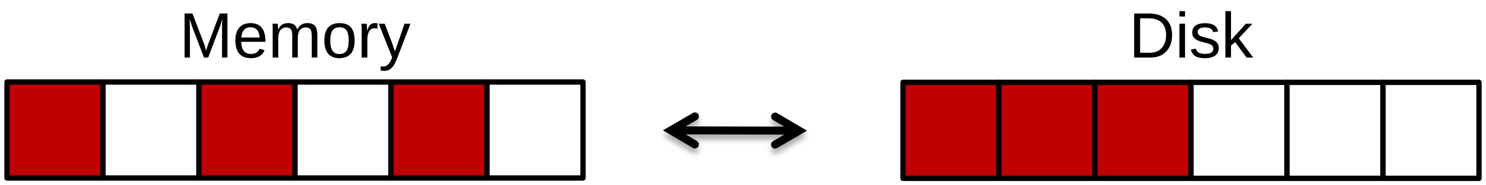


- Commonly, data access is contiguous in memory but noncontiguous on disk. For example, to reconstruct a global data structure via parallel I/O.

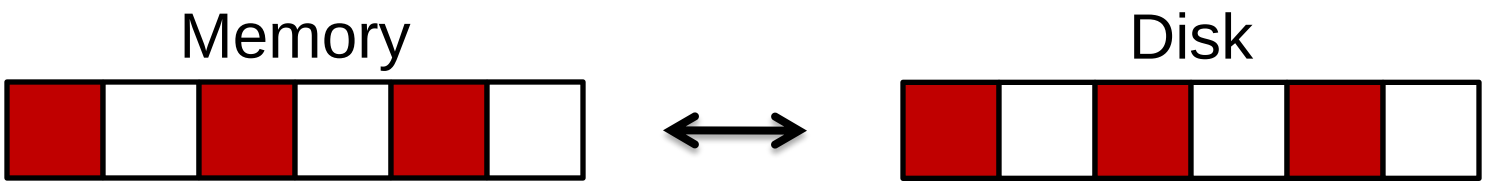


Data and Performance

- Sometimes, data access may be contiguous on disk but noncontiguous in memory. For example, writing out the interior of a domain without ghost cells.

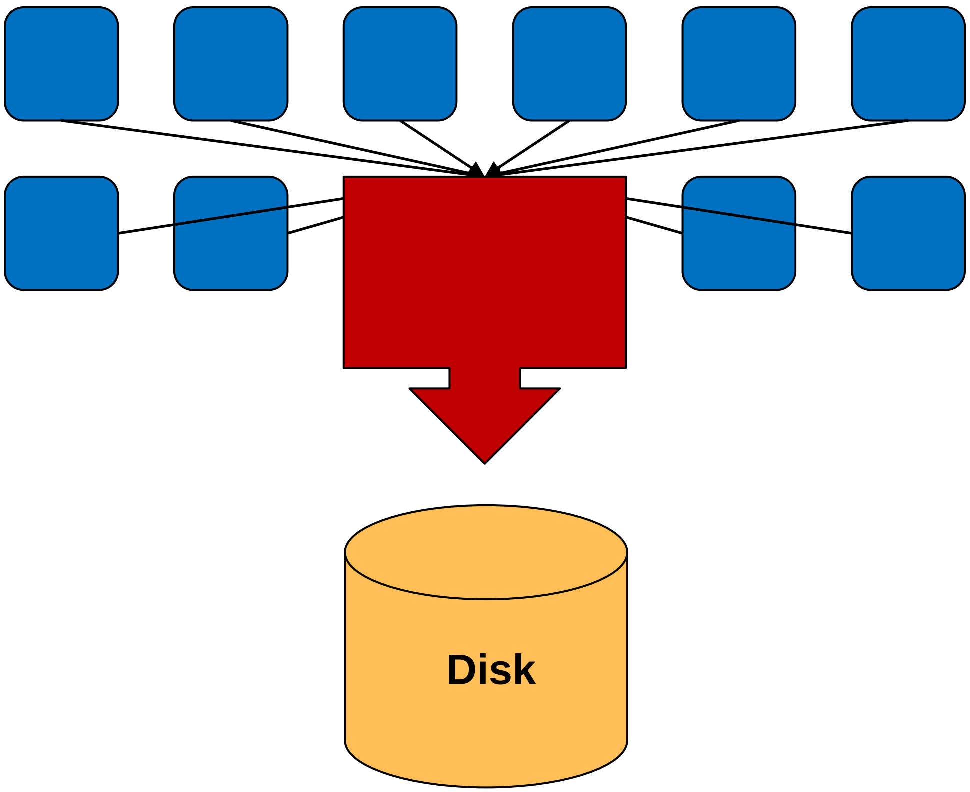


- A large impact on I/O performance would be observed if data access was noncontiguous both in memory and on disk.



Serial I/O: Spokesperson

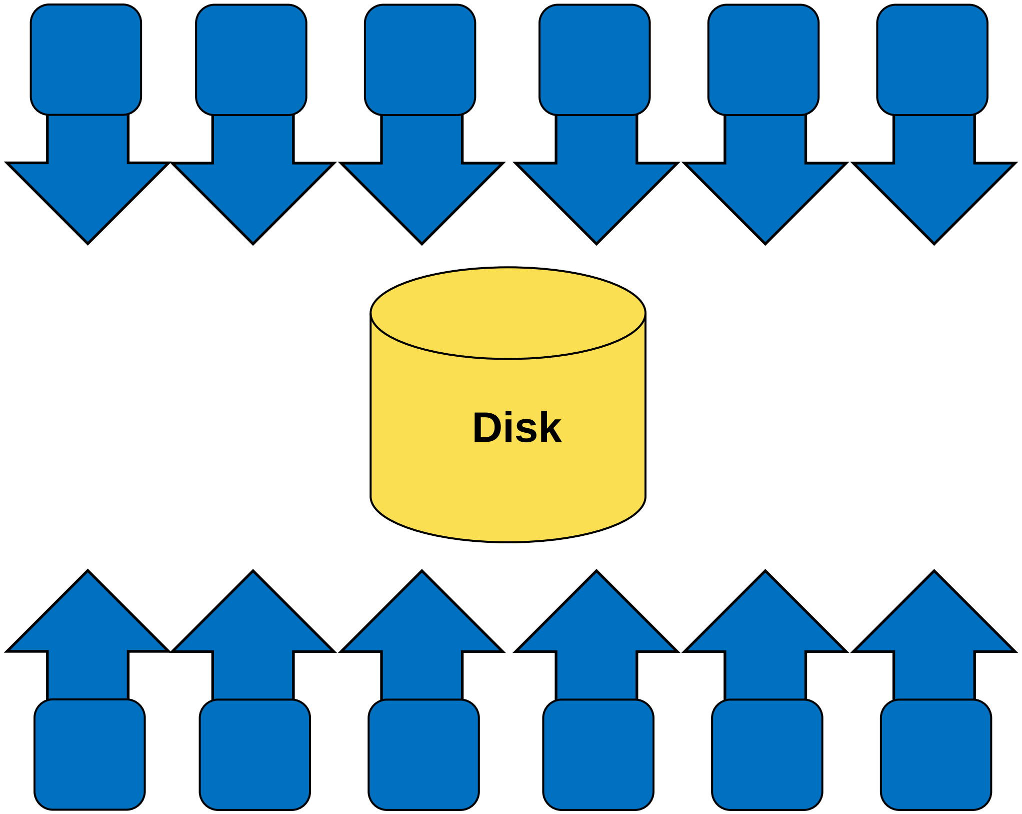
- Spokesperson
  - One process performs I/O.
    - Data Aggregation or Duplication
    - Limited by single I/O process.
  - Pattern does not scale.
    - Time increases linearly with amount of data.
    - Time increases with number of processes.



NICS

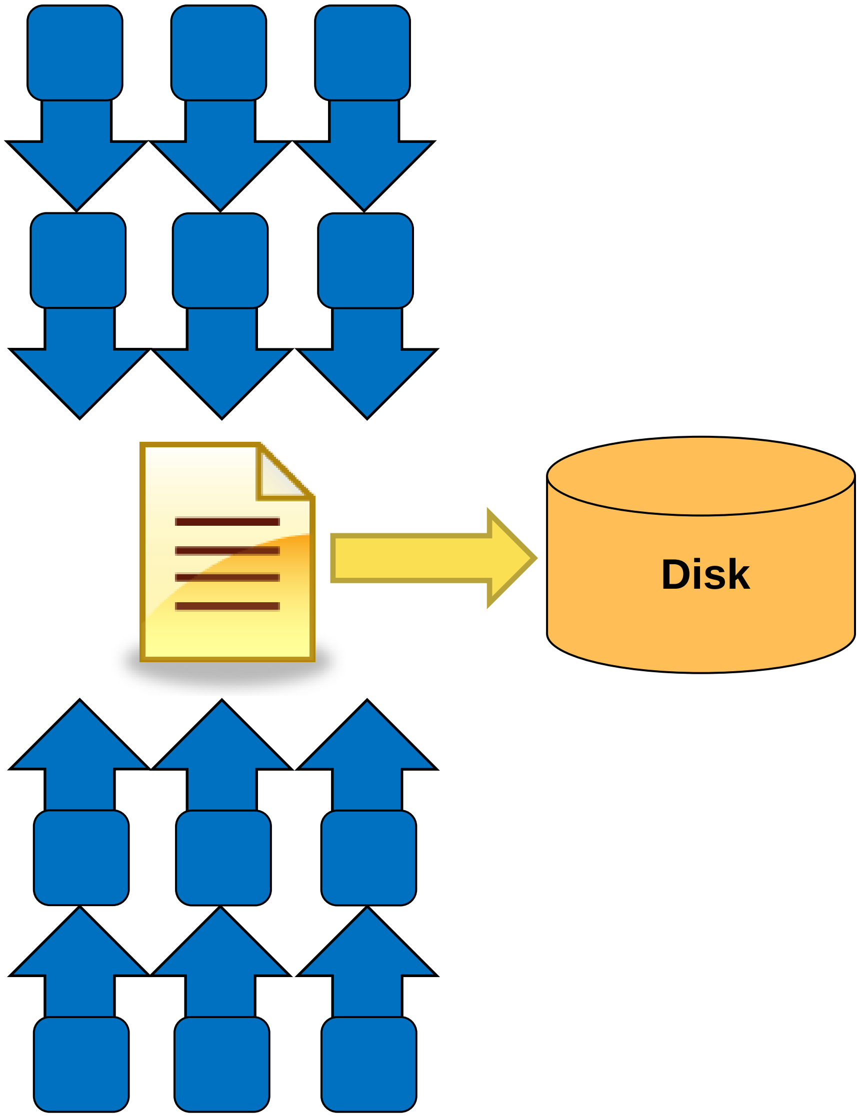
Parallel I/O: File-per-Process

- File per process
  - All processes perform I/O to individual files.
    - Limited by file system.
  - Pattern does not scale at large process counts.
    - Number of files creates bottleneck with metadata operations.
    - Number of simultaneous disk accesses creates contention for file system resources.



Parallel I/O: Shared File

- Shared File
  - Each process performs I/O to a single file which is shared.
  - Performance
    - Data layout within the shared file is very important.
    - At large process counts contention can build for file system resources.

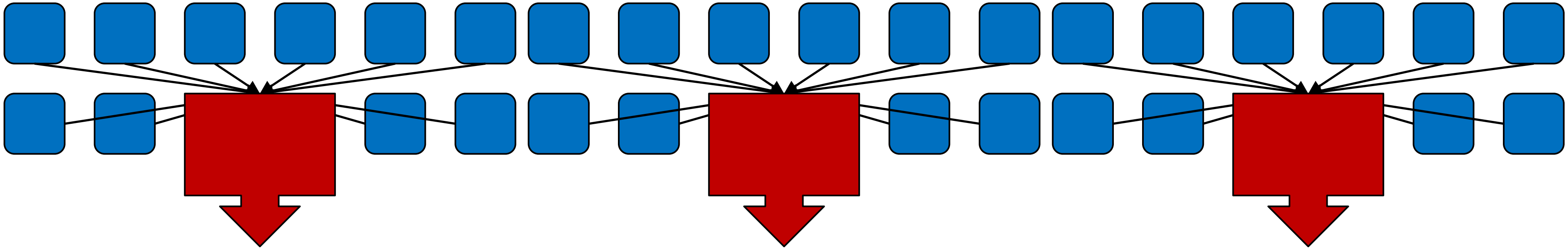


Pattern Combinations

- Subset of processes which perform I/O.
  - Aggregation of a group of processes data.
    - Serializes I/O in group.
  - I/O process may access independent files.
    - Limits the number of files accessed.
  - Group of processes perform parallel I/O to a shared file.
    - Increases the number of shared files

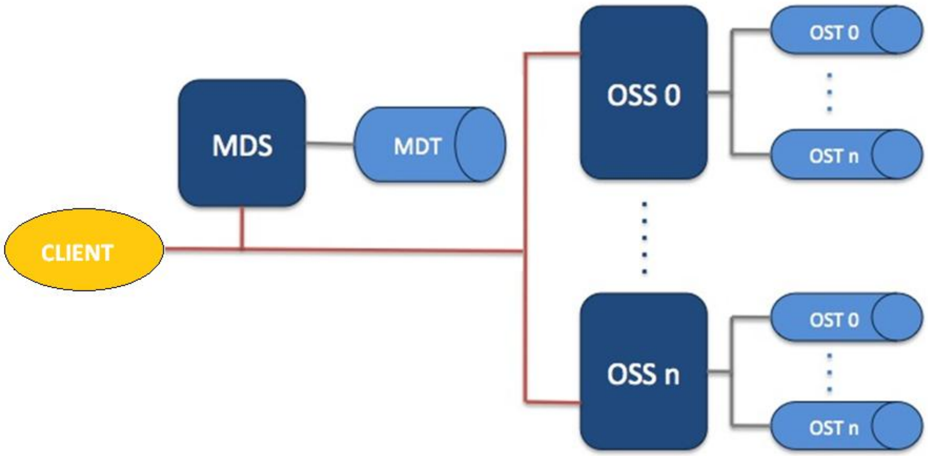
      → increase file system usage.
    - Decreases number of processes which access a shared file

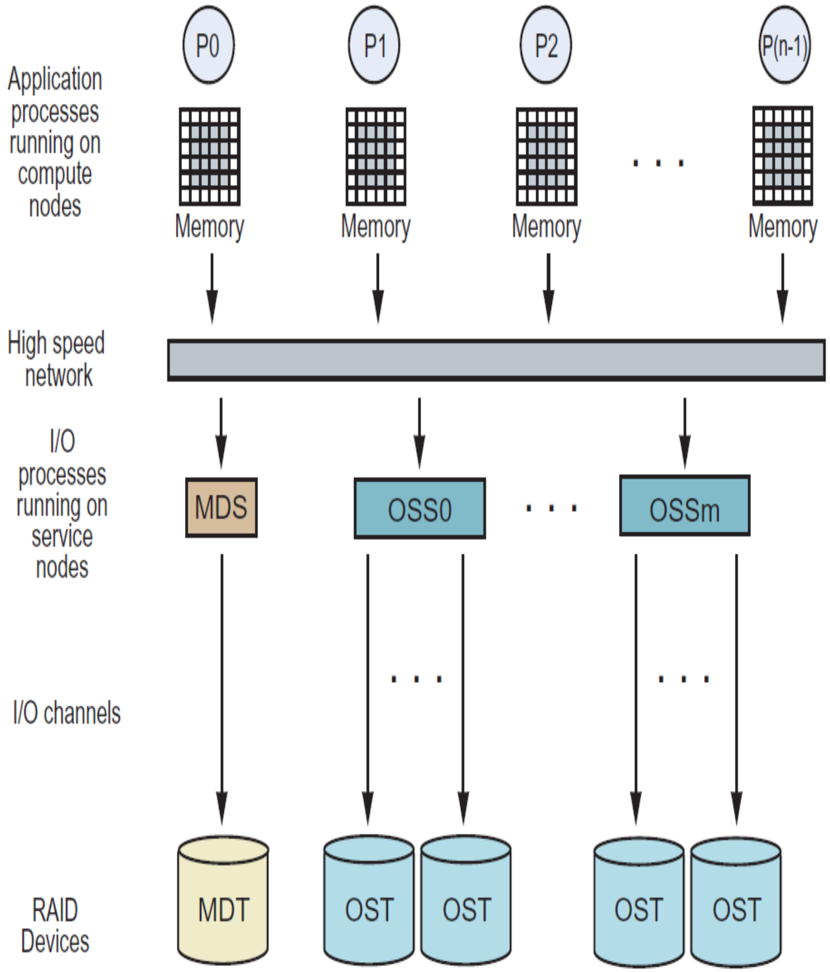
      → decrease file system contention.



File I/O: Lustre File System

- Metadata Server (MDS) makes metadata stored in the MDT(Metadata Target) available to Lustre clients.
- Each MDS manages the names and directories in the Lustre filesystem and provides network request handling for the MDT.
- Object Storage Server(OSS) provides file service, and network request handling for one or more local OSTs.
- Object Storage Target (OST) stores file data (chunks of files).





Striping: Storing a single file across multiple OSTs

- A single file may be stripped across one or more OSTs (chunks of the file will exist on more than one OST).

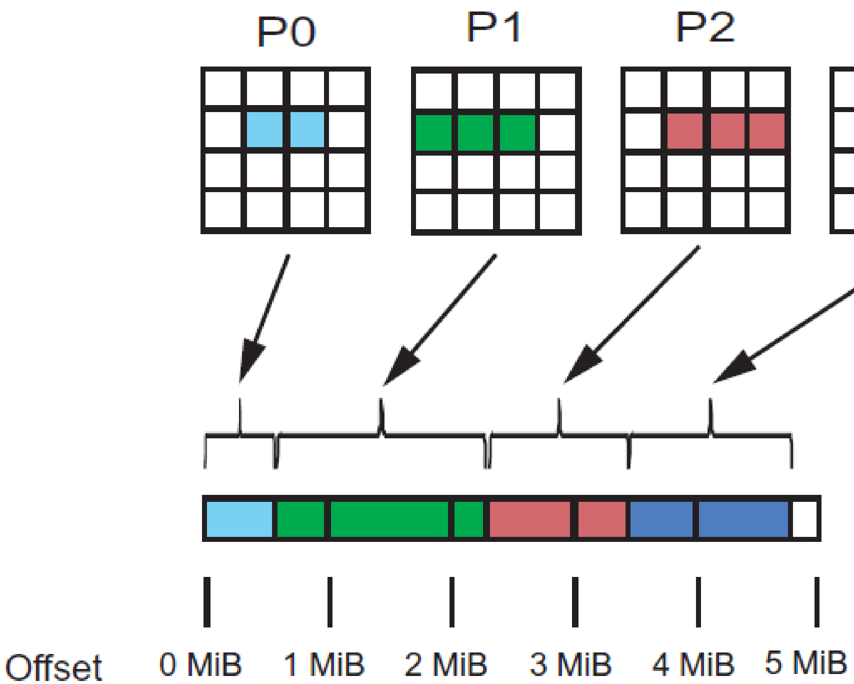
- **Advantages** :
  - an increase in the bandwidth available when accessing the file
  - an increase in the available disk space for storing the file.

- **Disadvantage**:
  - increased overhead due to network operations and server contention

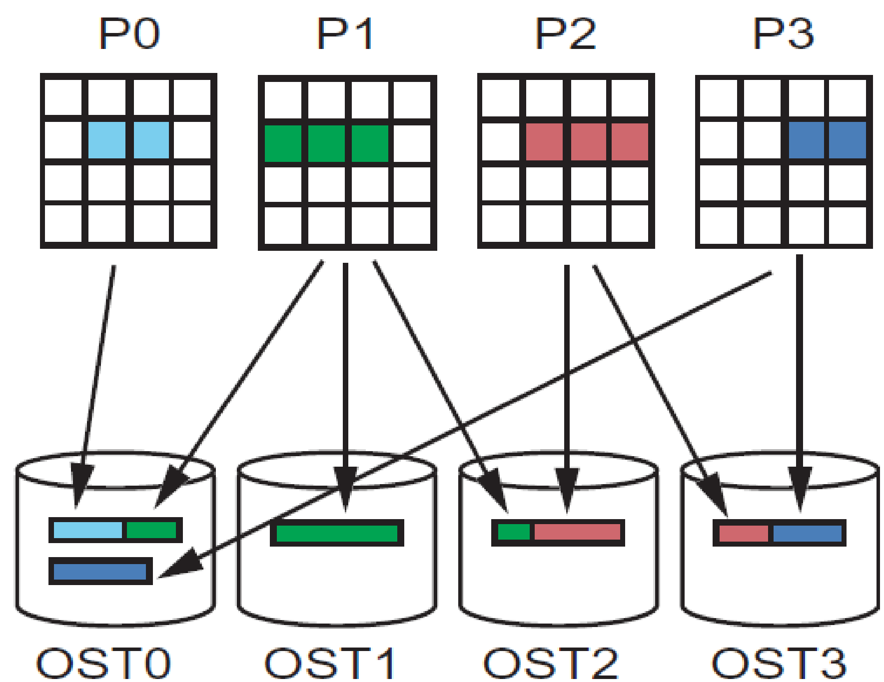
→ Lustre file system allows users to specify the striping policy for each file or directory of files using the lfs utility

NICS

File Striping: Physical and Logical Views



Four application processes write a variable amount of data sequentially within a shared file. This shared file is striped over 4 OSTs with 1 MB stripe sizes.



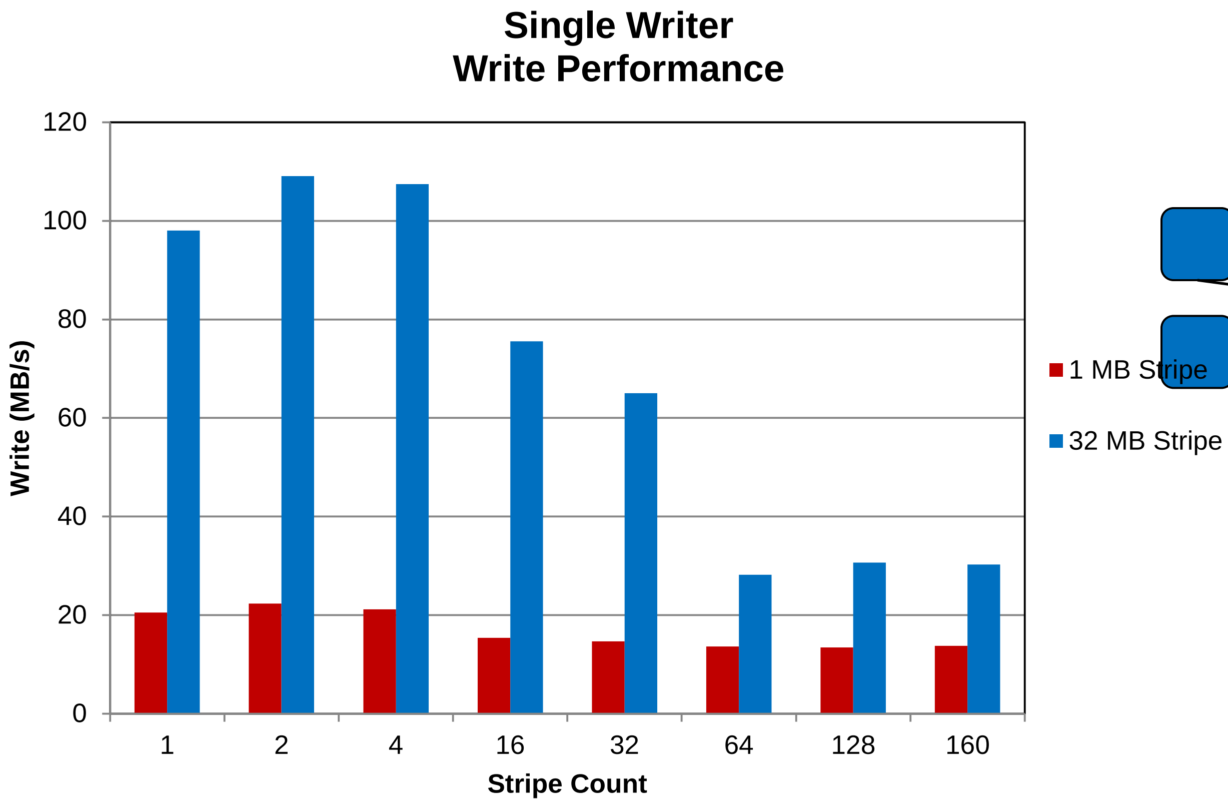
This write operation is not stripe aligned therefore some processes write their data to stripes used by other processes. Some stripes are accessed by more than one process

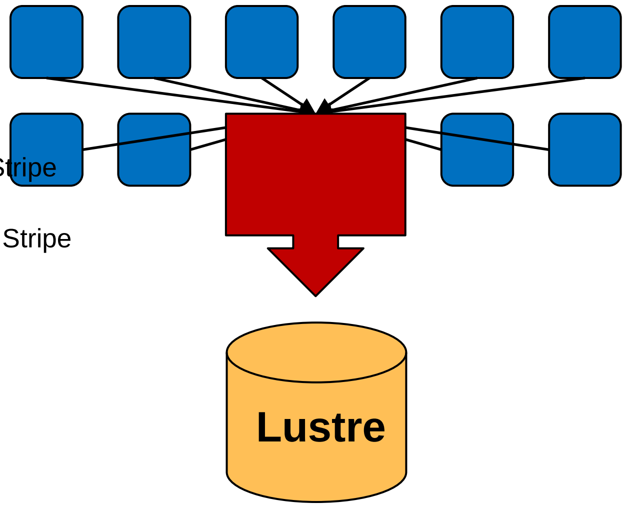
→ May cause contention !

OSTs are accessed by variable numbers of processes (3 OST0, 1 OST1, 2 OST2 and 2 OST3).

Single writer performance and Lustre

- 32 MB per OST (32 MB – 5 GB) and 32 MB Transfer Size
  - Unable to take advantage of file system parallelism
  - Access to multiple disks adds overhead which hurts performance



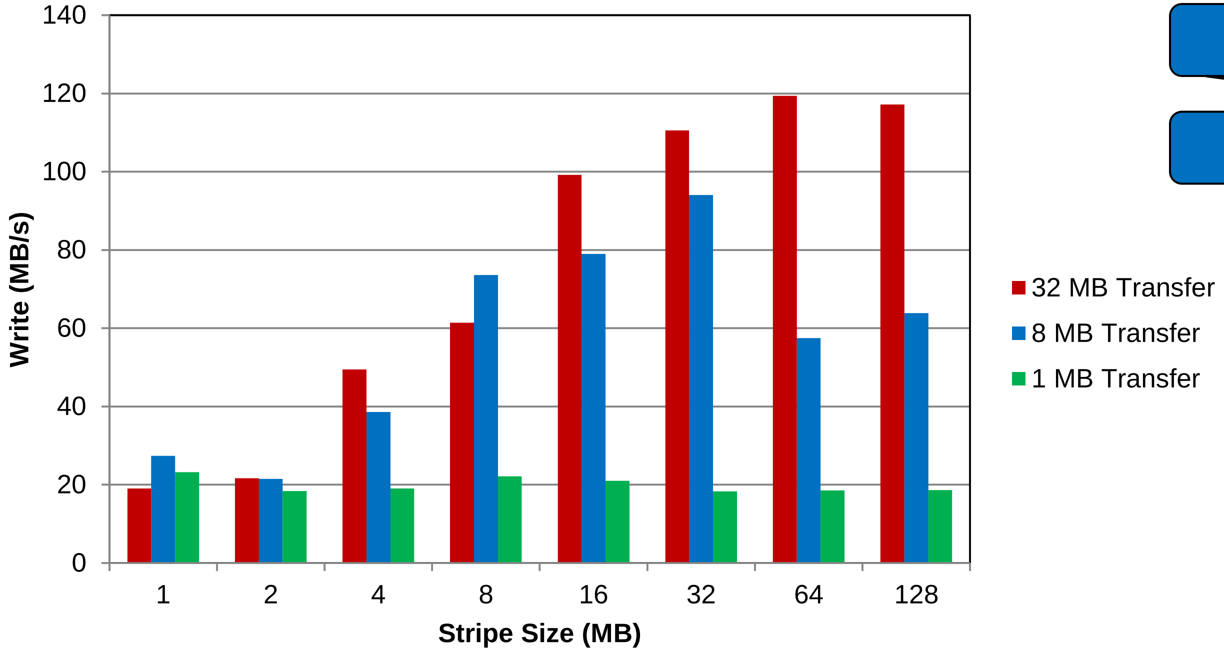


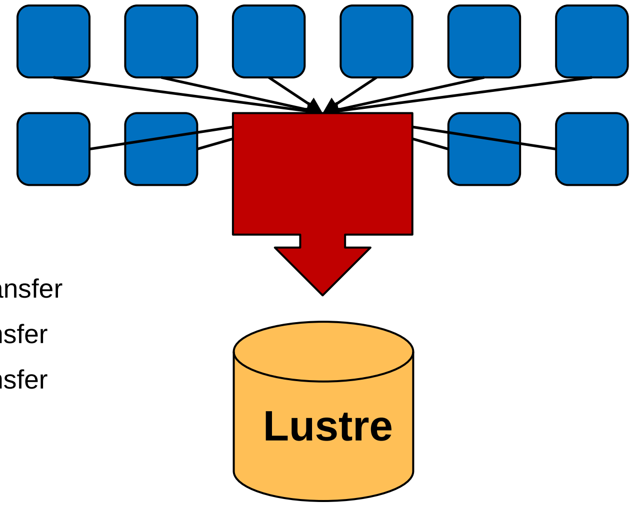
→ Using more OSTs does not increase write performance. (Parallelism in Lustre cannot be exploit)

Stripe size and I/O Operation size

- **Single OST, 256 MB File Size**
  - **Performance can be limited by the process (transfer size) or file system (stripe size). Either can become a limiting factor in write performance.**

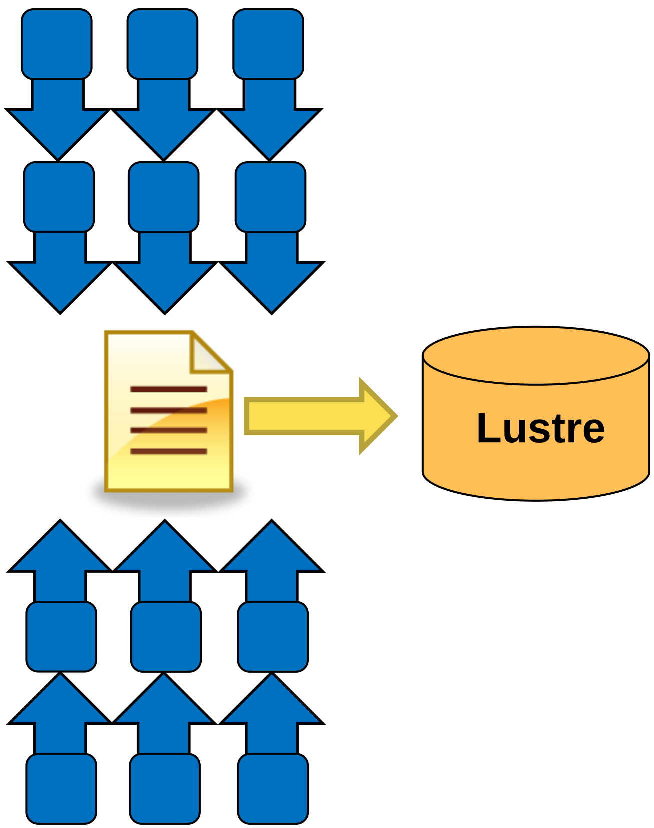
Single Writer
Transfer vs. Stripe Size





→ The best performance is obtained in each case when the I/O operation and stripe sizes are similar.
→ Larger I/O operations and matching Lustre stripe setting may improve performance (reduces the latency of I/O op.)

Single Shared Files and Lustre Stripes

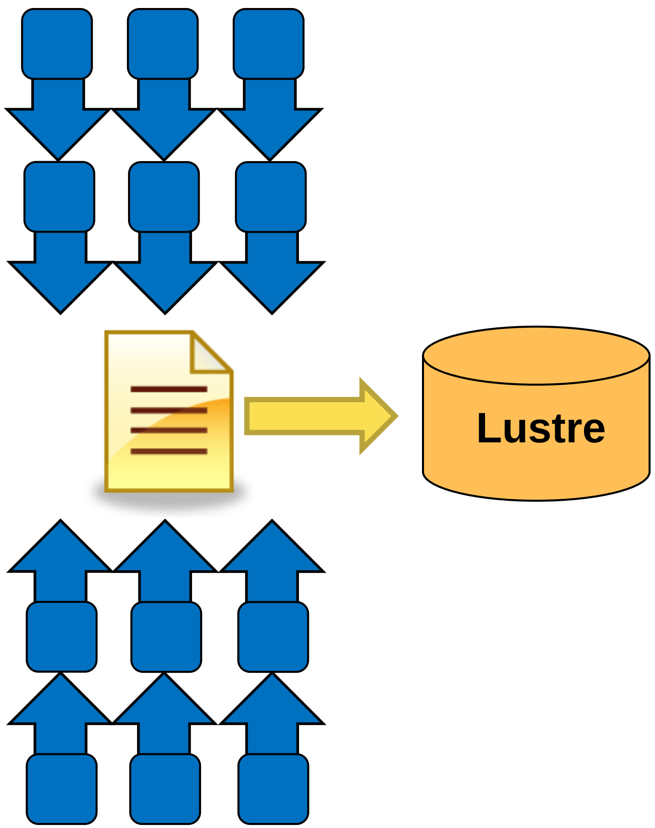


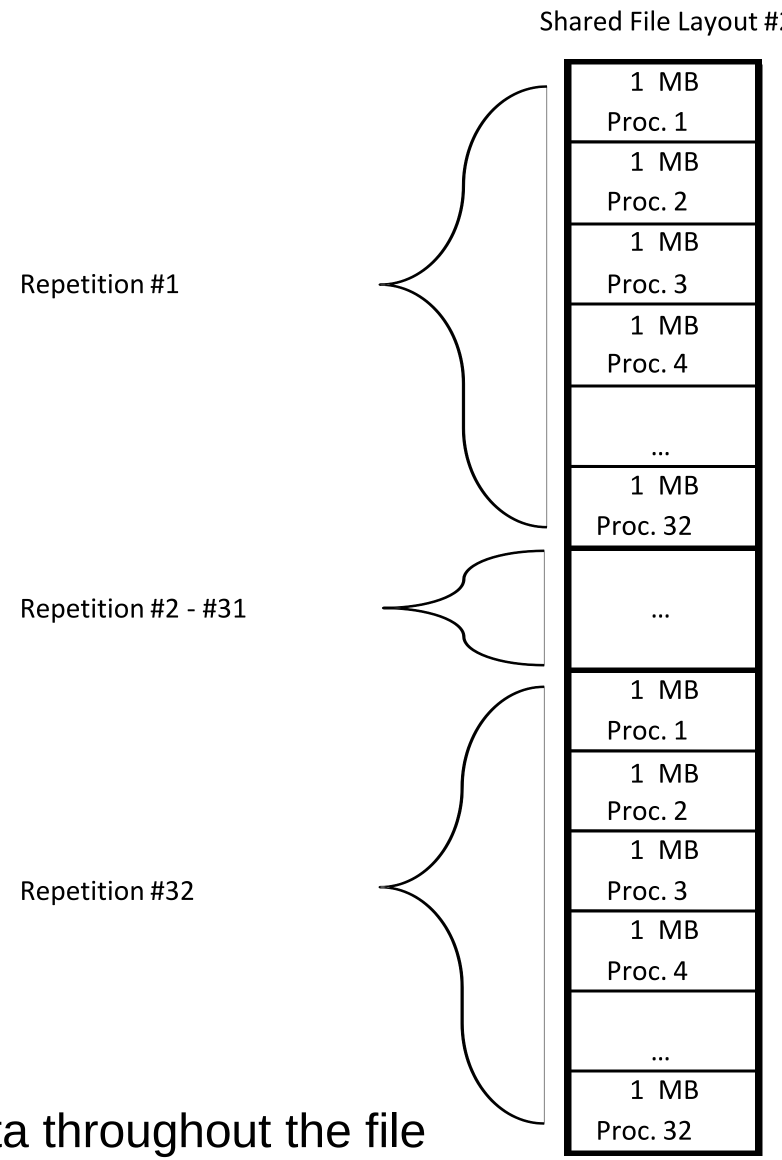
Shared File Layout #1

32 MB Proc. 1
32 MB Proc. 2
32 MB Proc. 3
32 MB Proc. 4
…
32 MB Proc. 32

Layout #1 keeps data from a process in a contiguous block

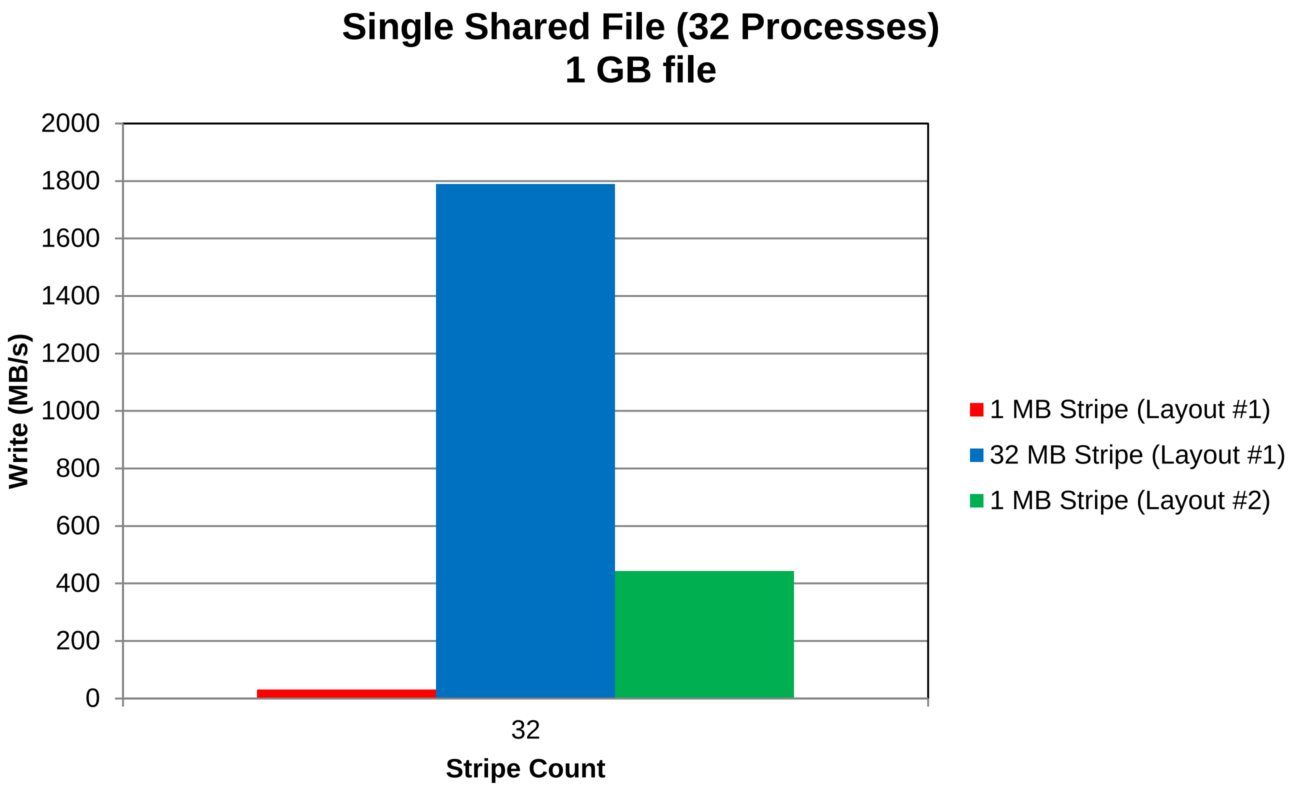
Single Shared Files and Lustre Stripes

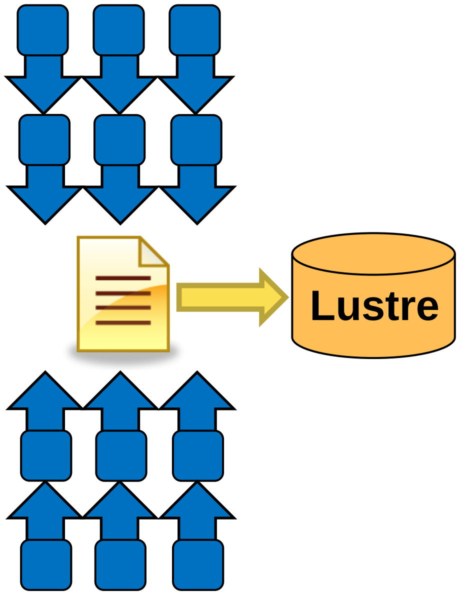




Layout #2 strides this data throughout the file

File Layout and Lustre Stripe Pattern





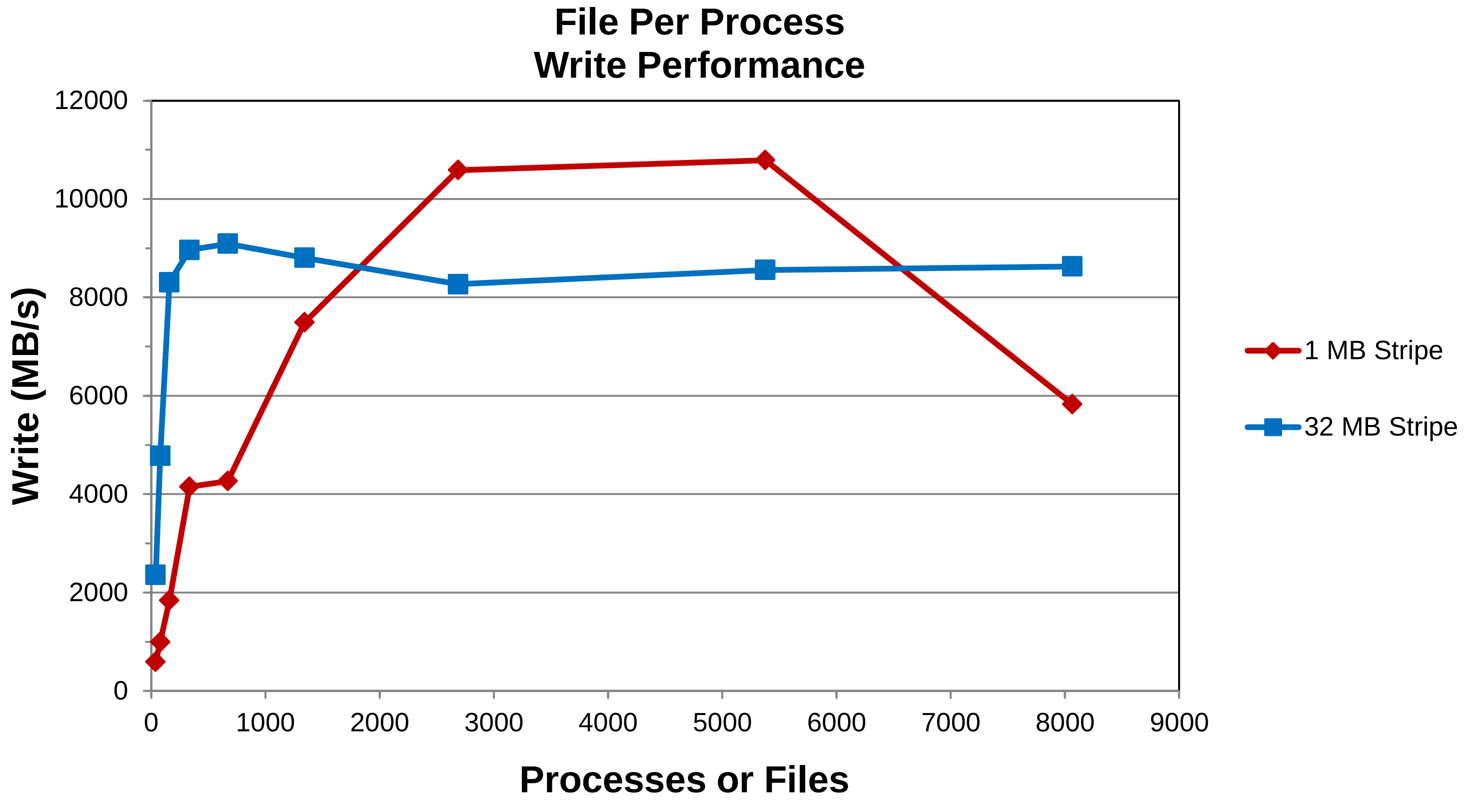
→ A 1 MB stripe size on Layout #1 results in the lowest performance due to OST contention. Each OST is accessed by every process.

→ The highest performance is seen from a 32 MB stripe size on Layout #1. Each OST is accessed by only one process.

→ A 1 MB stripe size gives better performance with Layout #2. Each OST is accessed by only one process. However, the overall performance is lower due to the increased latency in the write (smaller I/O operations).

Scalability: File Per Process

- 128 MB per file and a 32 MB Transfer size

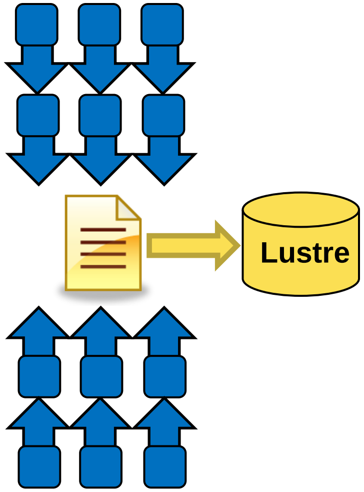


→ Performance increases as the number of processes/files increases until OST and metadata contention hinder performance improvements.

→ At large process counts (large number of files) metadata operations may hinder overall performance due to OSS and OST contention.

Case Study: Parallel I/O

- A particular code both reads and writes a 377 GB file. Runs on 6000 cores.
  - Total I/O volume (reads and writes) is 850 GB.
  - Utilizes parallel HDF5
- Default Stripe settings: count 4, size 1M, index -1.
  - 1800 s run time (~ 30 minutes)
- Stripe settings: count -1, size 1M, index -1.
  - 625 s run time (~ 10 minutes)
- Results
  - 66% decrease in run time.



NICS

Scalability

- Serial I/O
  - Is not scalable. Limited by single process which performs I/O.

- File per Process
  - Limited at large process/file counts by:
    - Metadata Operations
    - File System Contention

- Single Shared File
  - Limited at large process counts by file system contention.
  - File striping limitation of 160 OSTs in Lustre (on Kraken)

Outline

- Introduction to I/O
- Path from Application to File System
- Common I/O Considerations
  - I/O libraries
  - MPI I/O usage
  - Buffered I/O
- I/O Best Practices

I/O Libraries (MPI-IO)

- Many I/O libraries such as HDF5 and Parallel NetCDF are built atop MPI-IO.
- Such libraries are abstractions from MPI-IO.
- Such implementations allow for higher information propagation to MPI-IO (without user intervention).

MPI-IO Usage

- Included in the Cray MPT library.
- Environmental variable used to help MPI-IO optimize I/O performance.
  - setenv MPICH_MPIIO_HINTS
  - man mpi for more information
- If given appropriate information (stripe count, size) can choose aggregators in collective operations that are Lustre stripe aligned. (collective buffering).

NICS

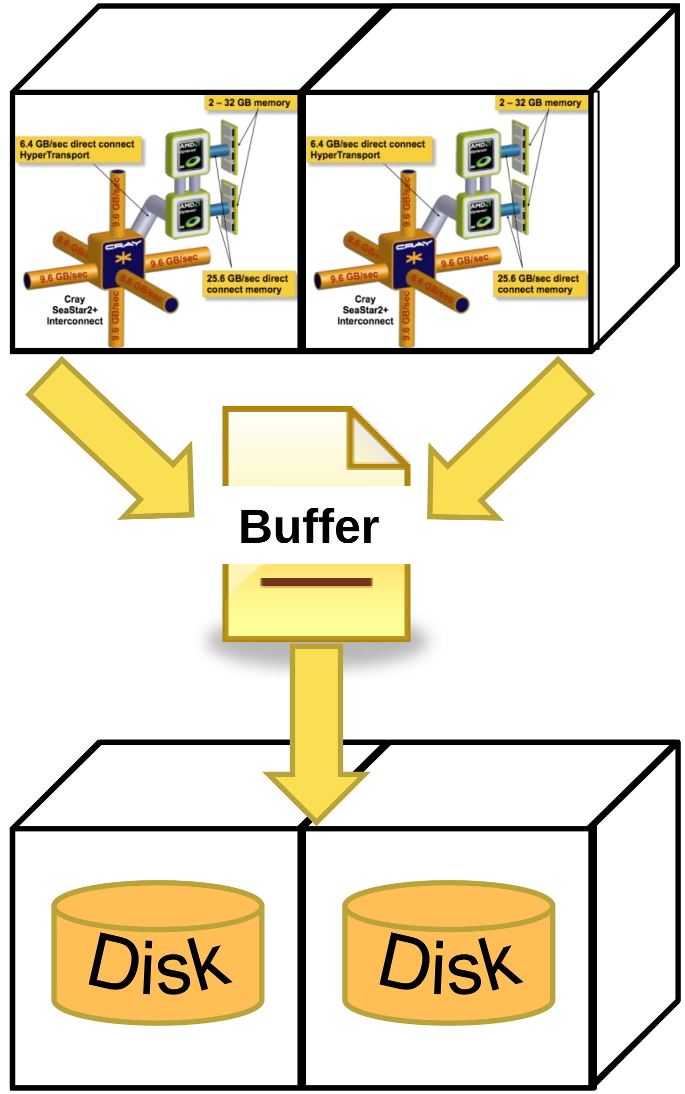
MPI-IO_HINTS

- MPI-IO are generally implementation specific. Below are options from the Cray XT5. (partial)
  - striping_factor (Lustre stripe count)
  - striping_unit (Lustre stripe size)
  - cb_buffer_size (Size of Collective buffering buffer)
  - cb_nodes (Number of aggregators for Collective buffering)
  - ind_rd_buffer_size (Size of Read buffer for Data sieving)
  - ind_wr_buffer_size (Size of Write buffer for Data sieving)

- MPI-IO Hints can be given to improve performance by supplying more information to the library. This information can provide the link between application and file system.

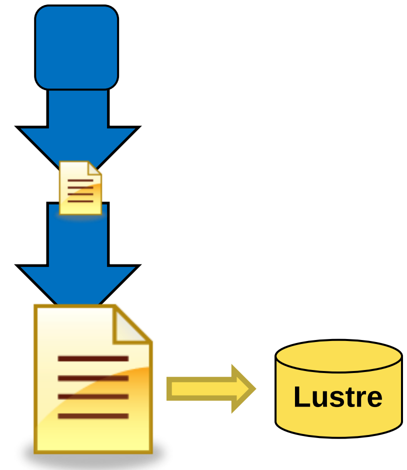
Buffered I/O

- **Advantages**
  - Aggregates smaller read/write operations into larger operations.
  - Examples: OS Kernel Buffer, MPI-IO Collective Buffering
- **Disadvantages**
  - Requires additional memory for the buffer.
  - Can tend to serialize I/O.
- **Caution**
  - Frequent buffer flushes can adversely affect performance.



Case Study: Buffered I/O

- A post processing application writes a 1GB file.
- This occurs from one writer, but occurs in many small write operations.
  - Takes 1080 s (~ 18 minutes) to complete.
- IO buffers were utilized to intercept these writes with 4 64 MB buffers.
  - Takes 4.5 s to complete. A 99.6% reduction in time.




```
File "ssef_cn_2008052600f000"
                     Calls         Seconds        Megabytes    Megabytes/sec    Avg Size
  Open                   1        0.001119
  Read                 217        0.247026         0.105957         0.428931         512
  Write            2083634        1.453222      1017.398927       700.098632         512
  Close                  1        0.220755
  Total            2083853        1.922122      1017.504884       529.365466         512
  Sys Read               6        0.655251       384.000000       586.035160    67108864
  Sys Write             17        3.848807      1081.145508       280.904052    66686072
  Buffers used               4 (256 MB)
  Prefetches                 6
  Preflushes                15
```

Outline

- Introduction to I/O
- Path from Application to File System
- Common I/O Considerations
- **I/O Best Practices**

I/O Best Practices

- **Read small, shared files from a single task**
  - Instead of reading a small file from every task, it is advisable to read the entire file from one task and broadcast the contents to all other tasks.

- **Small files (< 1 MB to 1 GB) accessed by a single process**
  - Set to a stripe count of 1.

- **Medium sized files (> 1 GB) accessed by a single process**
  - Set to utilize a stripe count of no more than 4.

- **Large files (>> 1 GB)**
  - set to a stripe count that would allow the file to be written to the Lustre file system.
  - The stripe count should be adjusted to a value larger than 4.
  - Such files should never be accessed by a serial I/O or file-per-process I/O pattern.

I/O Best Practices (2)

- Limit the number of files within a single directory
  - Incorporate additional directory structure
  - Set the Lustre stripe count of such directories which contain many small files to 1.

- Place small files on single OSTs
  - If only one process will read/write the file and the amount of data in the file is small (< 1 MB to 1 GB) , performance will be improved by limiting the file to a single OST on creation.

  → This can be done as shown below by: # lfs setstripe PathName -s 1m -i -1 -c 1

- Place directories containing many small files on single OSTs
  - If you are going to create many small files in a single directory, greater efficiency will be achieved if you have the directory default to 1 OST on creation

  →# lfs setstripe DirPathName -s 1m -i -1 -c 1

I/O Best Practices (3)

- **Avoid opening and closing files frequently**
  - Excessive overhead is created.

- **Use ls -l only where absolutely necessary**
  - Consider that “ls -l” must communicate with every OST that is assigned to a file being listed and this is done for every file listed; and so, is a very expensive operation. It also causes excessive overhead for other users. "ls" or "lfs find" are more efficient solutions.

- **Consider available I/O middleware libraries**
  - For large scale applications that are going to share large amounts of data, one way to improve performance is to use a middleware libary; such as ADIOS, HDF5, or MPI-IO.
  - On Kraken and Jaguar, I/O libraries are the third most used libraries at linking

Further Information

- NICS website
  - http://www.nics.tennessee.edu/I-O-Best-Practices
- Lustre Operations Manual
  - http://dlc.sun.com/pdf/821-0035-11/821-0035-11.pdf
- The NetCDF Tutorial
  - http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial.pdf
- Introduction to HDF5
  - http:// ww.hdfgroup.org/HDF5/doc/H5.intro.html

Further Information MPI-IO

- Rajeev Thakur, William Gropp, and Ewing Lusk, "A Case for Using MPI's Derived Datatypes to Improve I/O Performance," in *Proc. of SC98: High Performance Networking and Computing*, November 1998.
  - http://www.mcs.anl.gov/~thakur/dtype
- Rajeev Thakur, William Gropp, and Ewing Lusk, "Data Sieving and Collective I/O in ROMIO," in *Proc. of the 7th Symposium on the Frontiers of Massively Parallel Computation*, February 1999, pp. 182-189.
  - http://www.mcs.anl.gov/~thakur/papers/romio-coll.pdf
- Getting Started on MPI I/O, Cray Doc S–2490–40, December 2009.
  - http://docs.cray.com/books/S-2490-40/S-2490-40.pdf

Thank You !